Optimal Teaching Machines

Jacob Whitehill and Javier. R. Movellan

Abstract

Automated teaching systems have found success in various subjects areas, most notably high school mathematics [1] and introductory computer programming [2], but their full potential to transform modern education by providing each student with personalized instruction has not yet been realized. Most intelligent tutoring systems employ only impoverished sensors consisting of a keyboard and mouse. Not only do such sensors enforce a rigid interaction style which may be unnatural for the pupil, but because they ignore important real-time information about the student, they may also be achieving suboptimal learning gains. Employing more sophisticated sensors brings with it the new challenge of integrating *perception* and *action* in an intelligent manner, and principled approaches will be needed to tackle this problem. In this paper, we explore the idea that the Partially Observable Markov Decision Process (POMDP) may provide such a principled framework. We review the history of automated teaching systems since the 1960s to present day with an eye on how teaching decisions are made. We show that much of previous work on intelligent tutoring systems can be posed as a POMDP, and that the early research on teaching "m independent items" (e.g., vocabulary flashcards) from the 1960s and 1970s is relevant to contemporary "cognitive tutoring" systems. We further describe the computational limits of POMDPs for automated teaching scenarios, suggest ways in which these may be partially overcome, and discuss avenues for further research.

Index Terms

Intelligent Tutoring Systems, Partially Observable Markov Decision Processes, Stochastic Optimal Control, Machine Perception

I. Introduction

IT is well-known that one-to-one human tutoring is one of the most effective forms of instruction. Bloom [3] found that expert human tutoring can result in learning gains of two standard deviations above the mean score of students who were taught in conventional classroom style. Unfortunately, one-to-one human tutoring is also relatively expensive, and good tutors may not always be available. It is thus important to develop automated systems that can approximate the quality of human tutoring. While considerable progress has been made in automated teaching systems towards the elusive 2σ goal in a few domains (e.g., high school mathematics [1], introductory computer programming [2]), it is unclear whether the current heuristic approaches to making teaching decisions used by most automated tutors will generalize well to other teaching domains.

A striking feature of contemporary teaching systems is the unnatural, inflexible interaction style between human pupil and computer teacher that are typically enforced by these systems: With a few notable exceptions (e.g., Project LISTEN [4]), most contemporary tutors receive input from the student at only a few bits per second, using a standard keyboard and mouse, and this input is retrieved only at specific moments during the interaction. Not only is this unnatural for the human student, but by ignoring important cues about the student's state, it is also sub-optimal in terms of achieving learning gains. In contrast, human tutors and teachers continuously solicit feedback from their students, both consciously and subconsciously, over a variety of channels including speech, facial expression, eye gaze, and others. Recent years have seen dramatic improvements in sensor technology and machine perception algorithms which could allow automated systems to perceive through such channels, but adding new sensor inputs brings with it the new challenge of integrating rich, unstructured sensor inputs at varying time scales in an intelligent manner. For instance, what does it mean when a student looks away for a period of time when he/she should be solving a math problem? Is the student bored? Has he given up entirely? Or is he merely looking elsewhere while pondering the answer and about to deliver the solution? Knowing to what cause to attribute perceived behavior, and knowing how to respond to the underlying causes, poses new challenges for the field of automated teaching machines.

In this paper we explore the idea that the Partially Observable Markov Decision Process (POMDP) may provide a useful and principled approach to automated teaching, especially as the demands placed on the automated teacher by more sophisticated sensors continue to grow. POMDPs integrate noisy observations, state transitions, and reward signals in order to compute an optimal policy. As we will show, much of previous research on optimal teaching machines can be formulated in terms of the POMDP framework. By analyzing teaching scenarios from a POMDP perspective, one can benefit from new algorithms that are emerging from the POMDP and related research communities in order to solve previously intractable problems. In this paper, we review some of the most significant literature on automated teaching and discuss its relationship to POMDPs. We discuss the computational limits of the POMDP approach to optimal teaching, describe ways in which these limits can be partially overcome, and suggest important avenues for future research.

This paper analyzes optimal teaching from a somewhat theoretical and abstract perspective: Put loosely, we define teaching as *what* to teach to a student and *when*, in order to achieve a specific learning goal. We are not interested in the particular graphical user interface used by the teaching machine, or in the effort involved in entering the instructional content into the software, though these topics are of non-trivial importance. In the language of POMDPs, we wish to compute optimal policies for the agent (i.e., the teacher) so as to maximize some learning criterion in the environment (i.e., the student). From this abstract viewpoint, we examine three important scenarios: (1) Teaching m independent "items," where the notion of "item" is general and could be a single vocabulary word or perhaps a particular motor, perceptual, or cognitive skill; (2) teaching items that have knowledge dependencies, i.e., one item cannot be learned without knowledge of another; and (3) cognitive skill learning, i.e., how to perform a task such as solving an algebra problem.

The paper is structured as follows: We first provide a brief historical overview of the role of optimal control theory in automated teaching systems (Section II). We then describe the basics of Partially Observable Markov Decision Processes (Section III) and explain how they apply to the topic of optimal teaching (Section IV). Next, we describe three fundamental scenarios in automated instruction: m independent learning items (Section V), items with dependencies (Section VI), and cognitive skill acquisition using "production rules" (Section VII). It will turn out that the "m independent items" literature from the early years of automated teaching research may yet have bearing on much more recent cognitive tutoring systems. In Sections VIII, IX, and X, we discuss three issues that we consider to be the frontier of optimal teaching research: developing better learning models, integrating machine perception, and harnessing research from the machine learning and reinforcement learning communities. Finally, we make concluding remarks in Section XI.

II. Historical Perspective

The first endeavors towards automated teaching machines took place primarily at Stanford University in the 1960s and 1970s [5]–[10] and many of these efforts focused on teaching a list of "paired-associate" items, e.g., vocabulary words and basic facts. The decisions of which "item" to teach next were based on optimal control theory. The algorithms available at the time for solving these partially observable control problems were limited to exact-solution dynamic programming algorithms, whose computational time complexity is known to be doubly-exponential in the time horizon in the worst case. The teaching problems that were amenable to optimal policy computation were thus necessarily very small in scale, and it is thus not surprising that the optimality approach to computerized teachers mostly died off.

In the 1980s, the field of automated teaching was revived with John Anderson's "cognitive tutor" movement at Carnegie Mellon University. Cognitive tutors are based loosely on his ACT* and ACT-R theories of cognition [11], [12]. Notable examples of cognitive tutors include the LISP Tutor and Geometry Tutor [13]. Instead of teaching simple facts, these tutors provide students with structured practice environments in which to hone their proficiency in cognitive skills such as solving algebra problems and proving geometry theorems. Teaching decisions, such as which problem to present to the student next, are made mostly using heuristic methods.

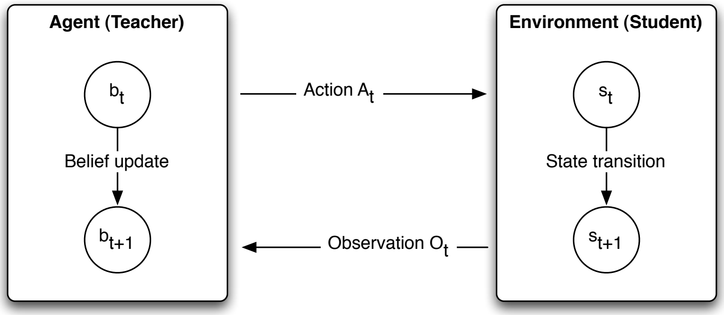


Fig. 1. Schematic of a POMDP.

Since the mid 2000s, there has emerged a small renaissance of the optimality approach to teaching, both for optimal inference and for decision-making: In [14], for example, Bayesian networks were used to optimally infer the student's problem-solving plan, and in [15] they were employed to assess whether students benefit from receiving hints during tutoring. For automatic hint generation, [16] and [17] employed fully observable MDPs to select hints so as to maximize the probability of the student reaching a solution. A few researchers have designed teaching systems that make decisions by maximizing some form of immediate reward [18], [19], i.e., one-step greedy look-ahead search over all possible actions. The only recent POMDP application to teaching of which we are aware is by [20], who investigate methods for decomposing a "monolithic" teaching POMDP into several smaller POMDPs, each one corresponding to a different type of student.

We believe that the recent re-visitation of the optimality approach to automated teaching can be significantly expanded to cover more aspects of decision-making and inference procedures, and also to integrate machine perception through more advanced sensors. In this paper we explore the utility for automated teaching of one particular framework for optimal control – the Partially Observable Markov Decision Process – which we introduce in the next section.

III. POMDPs

A Partially Observable Markov Decision Process (POMDP) is a probabilistic framework for sequential decision-making in which an *agent* (the teacher, in our case) interacts with the *environment* (the student) in order to maximize the long-term sum of accrued rewards (see Figure 1). In a POMDP, the agent cannot directly inspect the state of the environment, just as a teacher cannot peer into the brain of his/her students. Rather, the teacher must infer the state through noisy observations.

Formally, a POMDP consists of the following elements:

- A state space $\mathcal{S}$ that models the environment. The environment is assumed to be described by, and is Markovian in, its state. Due to the Markov property, the transition dynamics of the environment are independent of the past state given its current state and the action a_t:

$$P(s_{t+1} \mid a_t, s_t, s_{t-1}, \dots, s_1) = P(s_{t+1} \mid a_t, s_t)$$

  The state of the environment is not directly visible to the agent.
- A prior distribution $P(s_0)$ over the initial state of the environment.
- An action space $\mathcal{A}$ consisting of all actions the agent can execute and thereby affect the environment.
- An observation space $\mathcal{O}$ consisting of all possible observations the agent can make of the environment; this is the only information the agent receives.
- A reward function $R : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$ which specifies the value of the random variable R_t, the immediate reward received by the agent at time t when the agent takes action a_t and the environment is in state s_t.

- The state transition distribution $P(s_{t+1} \mid s_t, a_t)$ specifying the probability that the environment will transition from state s_t to state s_{t+1} when the agent takes action a_t.
- An observation distribution $P(o_t \mid s_t, a_t)$ specifying the probability the environment will emit observation o_t when in state s_t and the agent executes action a_t.
- A discount factor $\gamma \in [0, 1]$ specifying by how much immediate rewards are to be favored over future rewards.
- A horizon length T specifying the number of time steps for which the agent-environment interaction will take place. T can be infinite.

The agent can affect the environment by executing an action a at each time step. Based on the particular action executed and the current state of the environment, the agent receives an observation o from the environment. In a *partially observable* MDP, the observation is the only information the agent receives from the environment because the agent cannot observe the state directly. Along with the observation, the agent also receives a reward r at each time step, but in general the reward value is not observed by the agent[1]; hence, when making decisions, the agent can only maximize the *expected* sum of rewards given its actions and observations.

A. Computing an Optimal Policy

Given a model of the the environment state dynamics $P(s_{t+1} \mid s_t, a_t)$, observation (emission) probabilities $P(o_t \mid s_t, a_t)$, reward function $R(s_t, a_t)$, and prior state probabilities $P(s_0)$, the agent must devise a *policy* π that describes how it should act at each time step. Each policy induces a sum, over the entire time horizon, of expected discounted rewards:

$$V(\pi) = \sum_{t=0}^{T} \gamma^t E[R_t \mid \pi] \tag{1}$$

An *optimal policy* π^* is one that maximizes $V(\pi)$:

$$\pi^* = \arg\max_{\pi} V(\pi)$$

Note that an optimal policy seeks to maximize the expected sum of discounted rewards *over the long term*, until the end of the time horizon. This contrasts with other methods of decision-making, such as greedy one-step look-ahead search (see [18], [19], [21] for examples), which maximizes the reward *only at the next time-step*.

For computing a policy, the only information visible to the agent in a POMDP is the history of actions and observations:

$$H_t = \left((A_t, O_t), (A_{t-1}, O_{t-1}), \ldots, (A_1, O_1) \right)$$

This history can grow arbitrarily long, thus making it an unwieldy representation to store in memory. Fortunately, it turns out that the history can be compressed into a finite length representation without loss of relevant information for policy computation: Instead of defining a policy from the system history to actions, one can define the *belief* b_t, which is a vector whose ith component is defined as $b_{it} = P(S_t = i \mid h_t)$, representing the posterior distribution of the state at time t given the observed history h_t. The belief b_t is a sufficient statistic of the system history for maximizing V and allows the policy π to be formulated as a mapping from beliefs to actions.

Using the belief representation, one can re-cast the Partially Observable MDP of the environment's unknown *discrete* state into a Fully Observable MDP of the *continuous* belief state (see [22] for a derviation). The optimal policy can then be computed using, for example, the Value Iteration algorithm.

[1] More precisely, if the reward is observed and carries information about the state, then it *must* be incorporated as an observation because otherwise the agent would not be acting optimally.

B. Belief Updates

Given a policy π and the agent's current belief b_t about the environment's state s_t, the agent executes action $a_t = \pi(b_t)$ and then receives an observation from the environment o_t. The agent must now update its belief about the environment's state based on the newly acquired information. For discrete action and observation spaces, this can be computed efficiently using a recursion relation:

$$b_{t+1} \propto \sum_{s_t} P(s_{t+1} \mid s_t, a_t) P(o_t \mid s_t, a_t) b_t \tag{2}$$

After updating b_t to b_{t+1}, the agent then executes the next optimal action $\pi(b_{t+1})$, receives another observation, and so on, until the end of the session.

C. Horizon Length

A POMDP can assume either a finite horizon length T, in which case the interaction between agent and environment terminates after exactly T time steps, or an infinite horizon length. To ensure convergence in the general infinite-horizon case, rewards are typically discounted by a factor γ^t. Note that an infinite horizon does not imply that the agent has "infinite time" to act – due to the discount factor, the longer-term rewards will be given less weight than rewards that are received sooner. The main effect of an infinite horizon is that the optimal policy is *stationary*, i.e., it does not vary with time, which may be simpler to implement in some contexts.

D. Open Loop versus Closed Loop

A typical optimization problem consists of finding the optimal *closed loop* policy for a given POMDP, in which the belief, and hence the action, at time t is a function of both the previous actions and the observations. However, in principle the POMDP framework can also be used to compute an optimal *open loop* controller, whose actions from the beginning to the end of the session are pre-determined and are invariant to any observations. Open loop controllers in general deliver inferior performance to that of a closed loop policy, but they do have the advantage of simplicity, since nothing need be computed at run-time. They are also useful when observations are not available.

In this paper we focus mostly on closed loop control. We do discuss one research topic in which open loop policies are still being explored – optimal study schedules to exploit the Spacing Effect of Practice – in Section V-G.1.

E. Tractability

POMDPs are an elegant and powerful probabilistic framework to handle many sorts of control problems. The main difficulty encountered in using them is computational tractability in both the time and space sense.

The exact-solution value iteration algorithm of POMDPs alluded to above takes time $O(|S|^2|A|^{|O|^T})$, which is doubly exponential in the time horizon. However, using more recently developed *approximate* methods (e.g., *point-based methods*), the exponential dependency on time can be reduced to be linear. (The exponential dependency on number of observations still remains.) Another difficult challenge is the size of the state space, which can easily become very large, as we will describe.

IV. Teaching as a POMDP: A Simple Example

Here we present a simple example of how teaching can be formulated as a POMDP. While the teaching scenario may seem simplistic, it is fundamentally very similar to how acquisition of cognitive skills is modeled in cognitive tutoring systems (see Section VII). Suppose that a teacher is charged with teaching a student some skill. The student's knowledge of the skill is assumed to be binary (see Figure 2), i.e,

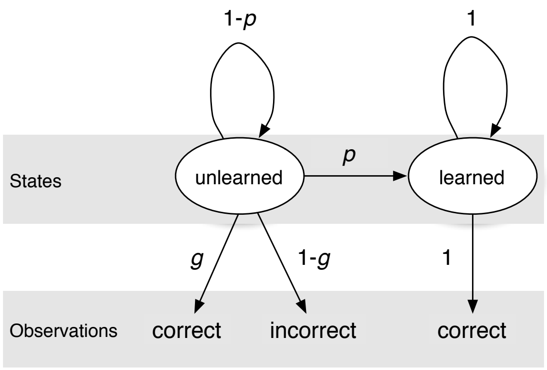


Fig. 2. The learning model of the Simple Example in Section IV.

it is either "learned" or "unlearned," as in [23]. The teacher can perform only one of three actions at each time-step: he/she can *teach*, meaning that the teacher attempts to transmit knowledge of the skill to the student without eliciting any feedback from the learner; he/she can *query* the student's knowledge by asking him/her to demonstrate the skill; and the teacher can *stop* the teaching session, after which no further *teach* or *query* actions can be performed. When the teacher *teaches*, the student's state may transition with probability p from the unlearned to the learned state. If the skill is already learned, then it will always stay learned, i.e., there is no "forgetting." When the teacher *queries*, the student's state is not affected, but the student will attempt to demonstrate the skill to the teacher, and the demonstration will either be *correct* or *incorrect*. If the skill is learned, then with probability 1 the student demonstrates the skill correctly. If the skill is unlearned, then the demonstration is correct with probability g (the "g" is for "guessing" in some contexts).

Associated with the *teach* and *query* actions are fixed costs – we assume negative rewards r_t and r_q, respectively. If the teacher *stops* and the student's knowledge state was *learned*, then there is a positive reward of r_s; otherwise, the reward for stopping is 0. For an example from daily life, the costs might consist of both the time that the teacher and student must invest, as well as any monetary costs, e.g., salary, teaching supplies, etc. Note the form that "querying" can play in modern education systems: standardized tests, for example, may retrieve valuable information about students' knowledge, but they also interrupt normal school instruction and hence incur a cost.

The question now is to determine what actions the teacher should take, and when. Forming such a set of decision rules is equivalent to computing a *policy* π for the teacher. In particular, we are interested in devising an *optimal policy*, i.e., a policy that maximizes $V(\pi)$ from Equation 1.

A. POMDP Formulation

The POMDP framework provides exactly the machinery necessary to find the optimal teaching policy for the above example. Let us call the unlearned and learned states u and l, respectively. Then, the entire

POMDP formulation is as follows:

$$\begin{aligned}
\mathcal{S} &= \{u, l\} \\
\mathcal{O} &= \{\text{correct}, \text{incorrect}\} \\
\mathcal{A} &= \{\text{teach}, \text{query}, \text{stop}\} \\
R(u, \text{teach}) &= r_t \\
R(l, \text{teach}) &= r_t \\
R(u, \text{query}) &= r_q \\
R(l, \text{query}) &= r_q \\
R(u, \text{stop}) &= 0 \\
R(l, \text{stop}) &= r_s \\
P(s_{t+1} \mid s_t, A_t = \text{teach}) &= \text{as shown in Figure 2} \\
P(s_{t+1} \mid s_t, A_t = \text{query}) &= 1 \text{ if } s_{t+1} = s_t; \\
& 0 \text{ otherwise} \\
P(s_{t+1} \mid s_t, A_t = \text{stop}) &= 1 \text{ if } s_{t+1} = s_t; \\
& 0 \text{ otherwise} \\
P(o_t \mid s_t, A_t = \text{teach}) &= 1 \text{ if } o_t = \text{correct}; \\
& 0 \text{ otherwise} \\
P(o_t \mid s_t, A_t = \text{query}) &= \text{as shown in Figure 2} \\
P(o_t \mid s_t, A_t = \text{stop}) &= 1 \text{ if } o_t = \text{correct}; \\
& 0 \text{ otherwise} \\
T &= \infty \\
\gamma &= 0.99
\end{aligned}$$

The particular parameter values we used were: $r_t = -1$, $r_q = -0.5$, and $r_s = 10$, $p = 0.2$, and $g = 0.1$. Note that, in our formulation above, the observation probabilities under the teach and stop actions are constant with respect to the state; the teacher always receives a "correct" observation and hence receives no information about the student's underlying state. Only when the teacher executes the query action does he/she receive meaningful information about the state.

B. Optimal Policy

Standard POMDP solution methods such as value iteration (see [22] for a detailed description) can be used to find the optimal policy. Value iteration computes $V^*(b_t)$, the expected long-term sum of rewards given the optimal policy π^* (see Equation 1) starting at time t. A plot of V^* and the associated optimal action for each belief b_t is shown in Figure 3 for $t = 1$. The x-axis represents $P(S_t = l)$.

As shown in the figure, when the teacher's estimate of the probability that the student has learned the skill is low (below about 0.35, in this case), then the optimal action is to teach. When the estimated probability of having learned the skill is above around 0.83, then the optimal action is to stop teaching since the student has probably already mastered the skill. In the remaining (middle) probability region, the teacher is very uncertain about whether the student is in state u or l. Teaching when the student is already in l would be a costly waste of time, but stopping when the student is still in state u would forgo the reward r_s. Since querying reduces uncertainty of the student's state and is less costly than teaching in our example, the optimal action for the middle region is to query.

This concludes our simple example of posing teaching as a POMDP. In the next sections we examine three fundamental teaching scenarios and how POMDPs can be used to find the optimal teaching policy for them. While they may seem simplistic, we believe they can model a large variety of teaching situations. The three scenarios we consider are m independent items, items with dependencies, and cognitive skill learning.

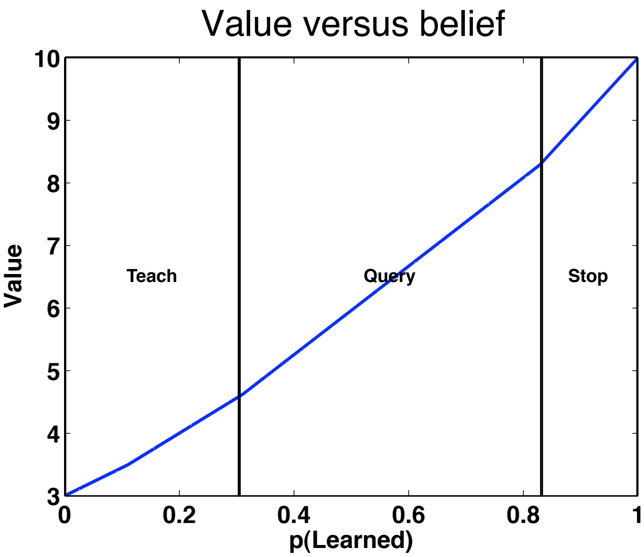


Fig. 3. Optimal policy π^* and optimal value function $V^*(b_t)$ for the example teaching scenario of Section IV.

V. Teaching m Independent Items

One of the fundamental scenarios in teaching is that there are m independent "items" to be taught. Such items could consist of vocabulary words, basic facts, or, as we will discuss later, cognitive skills. Given the action A_t and state $S_{i,t}$ of item i at time t, the state $S_{i,t+1}$ of item i at time $t+1$ is independent of $S_{j,t}$ for all items $j \neq i$. Knowing each item at the end of the teaching session is associated with some reward. The goal in the m independent items problem is to compute an optimal policy of which item to teach and how to teach it (if there are multiple alternative teaching methods) at each time t so as to maximize the expected discounted reward.

In our treatment of the m independent items problem, we assume that the teacher has a model of how the student learns. This assumption is important because it allows the teaching problem to be tackled using a POMDP in the first place. Other approaches are conceivable; for instance, one might forgo modeling the learning processes of the student and instead develop a policy which directly optimizes the student's performance on a test. In this paper, however, we assume that a learning model is available. We suggest a possible method of creating such models in Section VIII.

There are two main learning models in the literature that have been used for the m independent items problem: the *single-operator model* and the *all-or-nothing model*. As we will show, under certain conditions, the former is a special case of the latter, and the latter can be formulated directly as a POMDP. We discuss both these models below. Then, we describe the state-of-the-art of the specific kinds of m independent items problems whose optimal policies can be computed using POMDPs.

A. *Single-operator model*

One of the oldest learning models in the optimal teaching machine literature is the *single-operator model* [24]. The context is that, at each time step, the learning item in question is both taught and queried simultaneously. Under the model, the probability of receiving an *in*correct response decreases by a constant factor at each time step. Hence, if the item is taught at time t, then the probability of error at time $t+1$ is

$$P(e_{t+1}) = \alpha P(e_t) \tag{3}$$

where $\alpha \in [0, 1]$. Since Equation 3 is a linear difference equation, the model is sometimes also called the *linear model*.

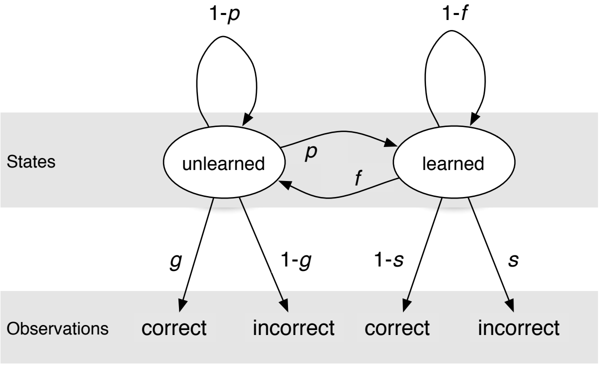


Fig. 4. The one-element model (OEM), also known as the all-or-nothing model.

The only parameters of this model are the decay rate of error, α, and the initial error probability $P(e_0)$, which is 1 minus the probability of a correct guess. Converting from a recursive to a closed-form solution,

$$P(e_t) = \alpha^t P(e_0)$$

The probability of correct response is then

$$P(c_t) = 1 - P(e_t) = 1 - \alpha^t P(e_0)$$

Two features of the linear model are noteworthy: First, the probability of incorrect response decays exponentially with teaching events. Second, the probability of correct/incorrect response at trial t is independent of any prior observations.

B. All-or-nothing model

Another popular learning model is the all-or-nothing model [23]. In contrast to the single-operator model, which is stateless, the all-or-nothing model is a finite-state machine with two states: unlearned u and learned l. When an item is in the unlearned state and the student's knowledge is queried by the teacher, then the learner can only guess the correct response with probability g. If, however, the learner is in the learned state for this item, then with probability 1 he/she responds correctly. The probability of transitioning from state u to state l is p. The model can also be formulated more generally to allow for "slipping" – the probability s that an item will be incorrectly recalled even when it is learned – and "forgetting" – the probability of transitioning from learned to unlearned. Usually, forgetting is only possible when the item is *not* taught at a specific time step. The complete model is depicted in Figure 4. Note the strong similarity to the POMDP example we presented in Figure 2 – though the all-or-nothing model predated the discovery of POMDPs, the all-or-nothing model is directly amenable to optimal policy computation using POMDPs.

Like the single-operator (linear) model, the all-or-nothing model also exhibits an exponential drop-off in probability of incorrect response. This can be readily seen from the fact that, assuming there is no slipping, an incorrect response can occur only if the item is still in state u at time t. Hence, the probability of incorrect response under the all-or-nothing model is

$$\begin{aligned}
P(e_t) &= P(O_t = \text{incorrect}) \\
&= P(O_t = \text{incorrect}, S_t = l) + P(O_t = \text{incorrect}, S_t = u) \\
&= P(S_t = u) P(O_t = \text{incorrect} \mid S_t = u) \\
&= (1-p)^t (1-g) \\
&= \alpha^t P(e_0)
\end{aligned}$$

Hence, if we set the decay rate of the single-operator model α equal to 1 minus the learning rate p, then the two models, in the absence of any observations, are equivalent. However, while the marginal distributions $P(e_t)$ are the same for both models, the joint distributions of states and observations over $t = 1, \ldots, T$ are very different. This results in different predictions of various learning statistics, including the average number of correct responses between successive errors, the average length of a run of incorrect responses, and more.

C. Multi-state Models

A natural extension to the all-or-nothing model is to add additional states. Empirically it was found [10], [25] that adding a state to represent "transient knowledge" (i.e., that can be forgotten) may fit experimental data more accurately than the two-state, all-or-nothing model. In a vocabulary teaching machine, [10] split the "learned" state l into two different states – "transient" t and "permanent" p. An item that is permanently learned is never forgotten, but an item that is only transiently learned may be forgotten.

D. Formulation as a POMDP

The above models deal with only one item at a time, but formulating the m independent items problem as a POMDP is a straightforward extension. Here we assume the all-or-nothing model as the model of learning. Given a set of possible teaching methods (independent of the particular item being taught) $\mathcal{T} = \{\text{teachMethod1}, \text{teachMethod2}, \ldots\}$ and a set of items $\mathcal{M} = \{1, \ldots, m\}$, then the set of actions $\mathcal{A} = \mathcal{T} \times \mathcal{M}$. A reward is given each time an item transitions from the unlearned to the learned state, and a negative reward is given for transitioning from learned to unlearned.

The most troublesome aspect of the POMDP for this problem is the state space. Using the simplest state representation, the number of states necessary to model the m independent items problem is 2^m because each item can be either learned or unlearned, and there are m such items. For standard value iteration solution methods, the probability of all possible joint states must be considered when calculating the optimal policy. Even though the transition function $P(s_{t+1} \mid a_t, s_t)$ for the joint state at time t to the joint state at time $t + 1$ factorizes into the product of the marginal distributions for each item, it is an open question how this fact can be exploited to reduce the computation needed to find the optimal policy.

The empirical effect of the exponential state-space representation is shown in the table below, which contains execution times of policy computation for $m = 1, \ldots, 5$ learning items on a 2.8 GHz Intel Pentium IV.

m	**Time (sec)**
1	.06
2	.10
3	.16
4	372.34
5	?

For $m = 5$, the memory requirements exceeded 32GB, rendering computation of the optimal policy effectively impossible on our machine.

To overcome the state space problem, POMDP solution methods that do not rely on value iteration and hence are not dependent on the state size, such as policy iteration (see Section X), may be useful. In addition, there are some important special cases of the m independent items problem that are tractable, which we discuss below.

E. Homogeneous Items

In the undiscounted, finite-horizon case where the rewards associated with learning each item are equal; where the transition and emission probabilities p, g, and s are equal across all items; and where there is no forgetting ($f = 0$), it can be shown [26] that the optimal policy under the all-or-nothing model is to teach the item whose current probability of being learned is a minimum over all items.

F. Undiscounted, infinite horizon

When a discount factor $\gamma < 1$ is imposed, or when the horizon length is finite, then $S_{i,t}$ and $S_{j,t}$ for two items $i \neq j$ cannot be truly independent: If item i is in the learned state, then that is evidence that time was spent teaching i; therefore, less time must have been spent teaching j, and the posterior probability of l for item j must necessarily be lower than if item i were unlearned. However, if the time horizon were infinite *and* no discount were imposed, then this dependency would be removed.

This intuition was exploited in [6] to prove that the optimal policy under the all-or-nothing model for teaching a set of m items can be decoupled into the optimal policies for teaching each item individually when $\gamma = 1$ and $T = \infty$. At a high level, each item is taught until teaching it is no longer worth the cost. The item is then set aside forever (i.e., the "stop" action for that item is executed), and the teacher moves on to the next item, which can be selected arbitrarily. Their proof assumes $f = 0$, but learning and emission probabilities can differ from item to item.

When the assumptions of infinite horizon and no discounting are justified, this result is very useful because it allows the POMDP of all m items to be decomposed into m component POMDPs. This reduces the computational complexity immensely.

G. Open Loop Case

The m independent items problem is one of the few domains where open loop teaching may be feasible. Since items are assumed to have no knowledge dependence structure, the teaching machine can reasonably "cycle through" the items (in some particular order) without concern for whether the dependencies of one item might have already been fulfilled by the learning states of others. The most general formulation of m independent item optimal teaching in open loop can be easily formulated as a POMDP. The observation space O is set to a single "null" observation which is always observed, regardless of learning state, and hence provides no information. Assuming there is only one method of teaching, then the action space is $\mathcal{A} = \{1, \ldots, m\}$, i.e., one action for each item. The all-or-nothing model, which gives rise to a state with 2^m possible values as described previously, is a natural if cumbersome representation in the POMDP. As explained before, when observations are ignored, the all-or-nothing model is equivalent to the single-operator model.

In its most general formulation, in which any item may be taught at any time, optimal policy computation for an open loop teacher is less difficult than closed loop policy computation since the observation space is of size 1, but still exponential in the number of time steps. One special case that can be solved analytically is when the items are assumed to be divided into equally sized "stacks" of size k, such that the items in the first stack are taught in round-robin order a fixed number of times n; the first stack is then set aside, the next stack is processed, and so on. Suppes [5] showed that, under a single-operator model including both learning and forgetting, the optimal stack size k is either m (i.e., the stack equals the entire set of items) or 1 (i.e., each stack is only 1 item, so that each item is taught n times in a row and then set aside) depending on whether the learning rate is higher than the forgetting rate, or vice-versa, respectively.

1) The Spacing Effect: Open loop optimal teaching is still an active field of research in psychology [27], especially in connection with the Spacing Effect [28], whereby the optimal length of time between multiple study sessions may vary depending on when testing will occur. Cepeda, et al [27] examined the question of the ideal inter-study time when the time until testing was longer than ever previously investigated. In their study, the gap between the second study session and the test (the "retention interval") was about 6 months. They found that lengthening the inter-study session gap initially causes the expected test performance to increase; after it reaches a maximum value, it then decays. Their study is the first to show that a study gap of greater than 1 day is optimal for a memory task of foreign language vocabulary, facts, and visual object names.

H. How Good is Optimal?

Given that, for large m, computing the optimal policy using standard POMDP methods can be intractable, it is important to assess how much of a gain an optimal policy can deliver compared to reasonable

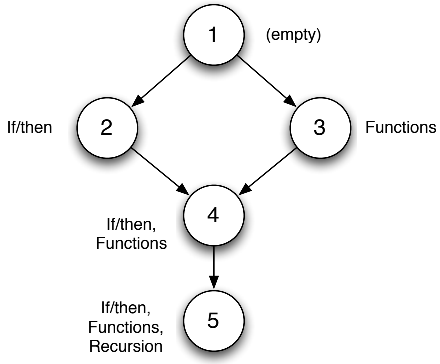


Fig. 5. An example of knowledge dependencies for the topic of programming recursion.

heuristic policies. We thus performed an experiment for a learning task in which m was small enough (we set $m = 4$) to allow optimal policy computation even with an exponentially large state space. We compared performance of the optimal policy (Optimal) to two reasonable alternatives: selecting the item whose current probability of being learned was minimal across all items (MinProb), and a simple round-robin selection (RoundRobin). The guessing and slipping parameters were fixed across all items: $g = 0.5$, $s = 0.1$. The learning and forgetting rates for the items were drawn uniformly at random from $[0, 1]$; hence, the items were *non*-homogeneous. We computed a non-stationary policy whose horizon length was the same as the simulation itself (40 time steps). No discount was imposed ($\gamma = 1$). One unit of reward was issued for each item that was in the learned state when the simulation ended. Each simulation was 40 time steps, and results were averaged over 50 simulations. The results are as follows ("return" is the sum of discounted rewards):

Method	**Mean Return (Std)**
Optimal	2.40 (0.57)
MinProb	1.46 (0.58)
RoundRobin	1.48 (0.68)

The difference between the optimal policy and a naive round robin policy was substantial – the optimal policy was nearly 60% better. The heuristic of selecting the item with minimum current probability turned out to be just as bad as round-robin. While this simulation is simplistic, it already suggests that optimal decision-making can result in real learning gains in teaching domains.

VI. Dependent Items

The m independent items scenario is useful for such situations as vocabulary learning, but it clearly fails to model many other learning tasks. Consider a student of introductory computer programming, for example, who is about to learn about recursion. A student must first know both what a function is, as well as an if/then statement (to distinguish between the base case and the recursive call), before he/she can master recursion itself. If we let item 1 be if/then statements, item 2 be functions, and item 3 be recursion, then the set of feasible knowledge states across all three items might be represented by Figure 5. Note that, due to the dependency, the number of states (5) is substantially less than the $2^m = 2^3 = 8$ that would be possible were the items independent.

The recursion example above is an example of a "hard" (as opposed to "soft") dependency because learning about recursion was assumed to be impossible if the student had not already mastered both the pre-requisites. This is a special case, however. More generally, a learning depedency exists if $P(s_{i,t+1} \mid s_{i,t}, a_t, s_{j,t}) \neq P(s_{i,t+1} \mid s_{i,t}, a_t)$ for some item $j \neq i$.

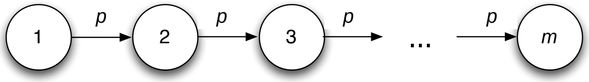


Fig. 6. State dynamics model of the Vygotsky Problem, in which the m learning items are completely dependent in the form of a chain.

A. Dependency Formulation under POMDPs

Defining a POMDP for learning items becomes more tedious as dependencies among items are introduced. Whereas for m independent items the transition probabilities between the 2^m states could be computed simply as the product of the marginals, with dependencies, more transition probabilities must potentially be somehow estimated from prior data. On the other hand, in the case of "hard" dependencies illustrated above, dependencies can sometimes also result in fewer states. To the extent that reducing the size of the state space reduces the computational cost of finding the optimal policy, it may be advantageous to exploit knowledge dependencies that exist in the target learning domain.

The optimal teacher of m *dependent* items is faced with a challenge not encountered when the items were independent. Whenever a knowledge dependency exists such that item i is required before item j can be learned, the teacher must first ascertain that the student knows i before attempting to teach j. An optimal policy, as computed with various standard POMDP solution methods, will implicitly query the student's state as needed to obtain high certainty that i has been mastered. We illustrate this with an example below.

B. The Vygotsky Problem

As an example of the most extreme set of dependencies among m items, consider the model depicted in Figure 6. The student's knowledge is modeled to be in a state $i \in \mathcal{M}$, where $\mathcal{M} = \{1, \ldots, m\}$ is an ordered set of states such that higher numbered states imply greater mastery of the subject matter. These items might, for example, correspond to a sequence of successively more difficult math problems, or perhaps even to the individual sentences of a lecture, each of which can be "learned" or "unlearned" by the end of the talk. The "Vygotsky Teaching Problem" (as we call it – he did not pose this problem himself) is to bring the student from his/her current state to the highest level possible within the given time length. A reward of 1 unit is given whenever the student advances from i to $i+1$. We assume that the state cannot regress, and that the state can advance by at most 1 level in each time step.

The action space is the set $\mathcal{A} = \{1, \ldots, m\}$. Teaching with action i is assumed to comprise two actions: first, a query at level i to see if the student has mastered that level; then, an attempt to advance the student to level i. If the level taught i is higher than the student's current level j, then the student can only guess the answer to the question, and the probability of correct guess is g. Otherwise ($i \leq j$), the student knows the answer and will respond correctly (with probability $1 - s$) unless he/she "slips" (probability s). Advancement to level i can occur (with probability p) only if the student is currently at level $i - 1$; otherwise, the transition probability is 0.

This notion of a critical zone within which learning occurs (only at the next higher level) is somewhat reminiscent of Vygotsky's Zone of Proximal Development (ZPD) [29]: The ZPD represents the range of difficulty of problems that a student can solve with or without an instructor's assistance. What the student can accomplish independently defines the lower end of the ZPD, whereas what the student can achieve with the aid of an instructor defines the upper end. According to Vygotsky, the role of education, then, is to provide learning experiences within a student's ZPD at each stage of his/her development. In accordance with this notion, we call the optimal teaching problem of this section the Vygotsky Problem.

Optimal Policy: In this problem, the m levels give rise to only m states; this is a marked contrast to m items which can each be in one of two possible states, which gives rise to 2^m states. We used value iteration to compute the optimal policy for the simplest case of the Vygotsky problem: $g = 0, s = 0, p = 1$, and the student's initial state is drawn uniformly at random from $\{1, \ldots, m\}$. The optimal teacher exhibits the following two-staged behavior:

1) The teacher locates the student's current state using approximately "binary search";
2) Once the teacher has identified the student's current state i (by finding the highest level at which the observation was "correct"), it proceeds to teach at level $i+1$ until another correct response is emitted. This implies the student is now at state $i+1$, and so the teacher then begins teaching at level $i+2$, and so on. This process repeats until either the time has run out or the student is at level m.

What is noteworthy from this example is that nothing in the model explicitly suggested that the teacher should first search for the student's current state, let alone perform the optimal search algorithm (binary search). It simply emerged from the model and the reward structure that this behavior was optimal. While the policy for this particular example might have easily been developed by hand, the point of principled frameworks such as POMDPs is that *it didn't have to be* – the inference procedure deduces the optimal behavior automatically.

VII. Cognitive Skill Learning

Up to now, in our discussion of optimal teaching we have treated knowledge as a set of *facts* (items) which can be either known or unknown. What has been missing is how learning a set of items might enable a person to *do* something, such as prove a geometry theorem or solve an algebra problem. Cognitive theories such as the Adaptive Character of Thought frameworks (e.g., [11], [12]) help to fill this gap by defining cognitive skills as the collection of independent *production rules*. Arguably, the most influential automated teaching systems [13] to date have been inspired by ACT-R. As we will show, cognitive skill learning under production systems such as ACT-R reduces to the learning of m independent items.

ACT-R was developed by John Anderson [12] as the culmination of several earlier models of cognition (e.g., ACT, ACTE, ACT* [30]). It models the memory and decision-making processes of human cognition as a *production rule system*. In such systems, a given task is decomposed into a sequence of subgoals using *production rules*. Production rules are if/then statements that contain a *pre-condition*, or required state of working memory for the rule to *fire*; as well as a *consequence*, or action associated with that rule firing.

The ACT family of models of cognition has inspired probably the most influential automated teaching machines to date, including the Geometry Tutor [13], the LISP Tutor [13], and Andes [31], a physics tutor. These systems are commonly referred to as "cognitive tutors". By 2009, the array of cognitive mathematics tutors had been deployed more widely than perhaps any other teaching machine, in over 2600 schools among 500,000 students [32]. The educational gains of these tutors is well established, with mean score improvement over students without automated tutoring of at least one standard deviation [2]. It is thus crucial to understand precisely how ACT-R and the cognitive tutoring systems themselves operate.

Unfortunately, obtaining a full grasp of ACT-R and the cognitive tutoring systems is problematic: the ACT-R literature, which is distributed across a vast collection of journal articles and monographs, is marred by a lack of specificity and consistency. Important parts of the definition vary from source to source, with no explanation as to why the change was made, or even an acknowledgement that the change had occurred at all. (An example of this is the decay rate of a learning event for estimating its effect on activation.) Mathematically precise terminology such as "log odds of X" (for some event X) is used informally, without any proof that the associated quantity equals what it should. However, ACT-R is also ingenious in how it decomposes complex cognitive skills (e.g., proving geometry theorems) into the collection of individual elements ("production rules") that can be learned independently.

Below we summarize the main contributions of the ACT-R model as it pertains to automated tutoring systems. As a concrete anchor point for how ACT-R is used in practice, we will examine a demonstration version of the Fraction Tutor [33]. While this tutoring system is simplistic, the fact that it is open source and hence its production rules can be inspected directly offers an invaluable opportunity to understand how a cognitive tutor is actually implemented in practice.

A. Declarative versus Procedural Knowledge

Under ACT-R, human knowledge is divided into *declarative* and *procedural* knowledge. *Declarative* knowledge comprises many *knowledge chunks*, which are analogous to variables in a computer program. Variables can take on values, but they do not directly perform any action. Examples of chunks include such statements as $x = 0$ (for an algebra problem), $\overline{AB} \perp \overline{BC}$ (in a geometry proof), and "the current goal is to write a recursive function that calculates the factorial of x." The full ACT-R theory models the strength of memory of knowledge chunks using "activation" [11], [12], [30], which supposedly predicts both latency and probability of recall of that chunk. Recent work [34] applied this theory to the "m independent items" problem by attempting to maximize gain in chunk activation using greedy one-step look-ahead for item selection.

Procedural knowledge consists of the set of *production rules* that are known by the student. Production rules are if/then statements that specify how a particular goal can be achieved when a specified pre-condition is met. Two typical production rules might be, "If the goal is to prove triangle similarity, then prove that any two pairs of angles are congruent", and "If the goal is to solve for x AND the equation $x + a = c$ is currently in declarative memory, then subtract a from both sides."

B. Solving Problems under ACT-R

Under ACT-R, the cognitive skills necessary to solve math problems, program a computer, understand physics, etc., are modeled as nothing more than a collection of production rules that the student must master. The essence of cognitive tutors is to help students learn these rules by providing them with *practice opportunities*: the tutor selects new problems containing production rules the student has presumably not yet mastered. The system designer of the cognitive tutor must define the set of important production rules a priori.

When the student tackles a specific problem within the tutor, he/she implicitly fires production rules, thereby affecting state of declarative memory. These changes to working memory may then cause another production rule to fire, and so on, until the problem is solved. The "snapshots" of the current set of knowledge chunks in declarative memory correspond to the state of the student, represented by nodes in a solution graph (described below). The production rules that fire and thereby alter the state of declarative memory are the edges between the nodes.

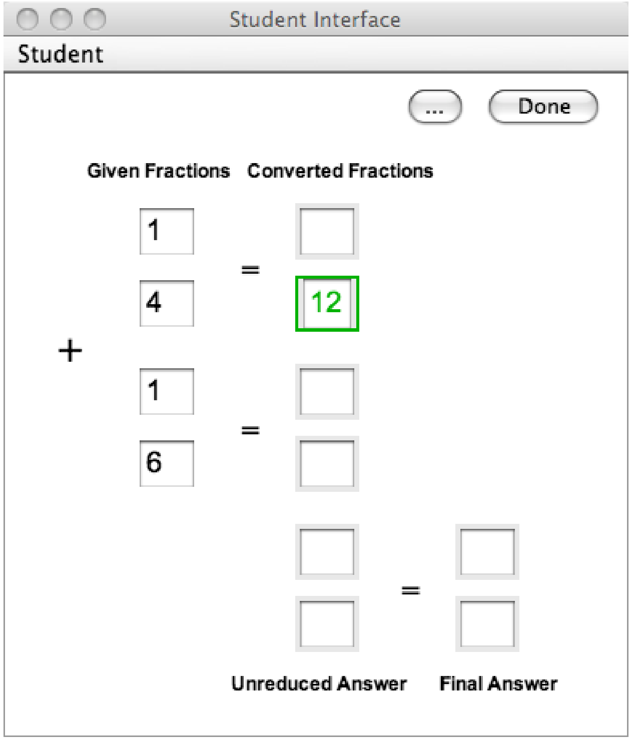
C. Example: Fraction Tutor

To make the ideas described above more concrete, let us consider the setting of teaching students to manipulate fractions. Suppose the task is to add two fractions with possibly differing denominators. The student must first find a common denominator for the two fractions, convert the numerators, add them, and then reduce the combined fraction to the lowest possible denominator. This is exactly the task that is taught by the Fraction tutor, whose graphical interface is shown in Figure 7, by providing the student with opportunities to practice this skill. The system designer of the Fraction Tutor must compose a list of all possible ways in which the problem could be tackled by the student, and then model each method using appropriate production rules. In the fraction addition example, a student could reasonably either find the least common denominator (LCD) of the two fractions, or she could also choose to simply multiply the two denominators. Each of these two actions is modeled as a different production rule:

P1 : IF the goal is to add two fractions, THEN set as a subgoal to find the LCD.

P2 : IF the goal is to add two fractions, THEN set as a subgoal to multiply the two denominators.

The designer might also wish to model a "buggy" strategy such as the erroneous step of adding the two numerators and denominators separately. Modeling this mistake as a buggy production will allow the cognitive tutor to catch this error and inform the student of his/her mistake.



1
4
12
1
6

Fig. 7. Graphical user interface of the Fraction Tutor. At the current time step, the student has correctly computed the least common denominator of the two fractions and entered this denominator into the appropriate textbox. Since this input is correct, it is shown in green.

D. Model Tracing

The set of production rules defined by the system designer induces a solution graph (see Figure 8) of all possible solutions to the given problem posed to the student (in fact, the figure includes only a few representative paths). For small sets of production rules, this graph can be computed automatically by theorem proving software. At each point in time while using the tutor, a student may be at any node within this graph. To keep track of the student's current state, the cognitive tutor makes use of observations, described below:

In the Fraction Tutor, not only can a production rule set new subgoals and add chunks to memory, but it can also trigger actions by the student which are then observed by the teacher. In this way, production rules define a *generative model* of how observations originate. For instance, one production rule used in the Fraction Tutor is of the following form:

P3 : IF the goal is to add fractions AND they have a common denominator d AND their component numerators are n_1 and n_2, THEN fill in the numerator part of the answer textbox with value (n_1+n_2).

Another production rule might be similar except that the observation would be to write the common denominator d in the answer textbox instead. Given the actual student action observed by the cognitive tutor at the current time step, the tutor can infer which production rule must have fired. The process of updating one's belief about which productions fired and which state the student currently inhabits is known as *model tracing*. In the language of POMDPs, it corresponds to a simple belief update.

E. Hints

The Fraction Tutor as we have described it thus far only monitors the student's progress. Upon request from the student, the tutor can also provide a hint. A hint can be attached to each production rule and suggests a reasonable course of action from each node in the solution graph. For instance, at the start state, the hint might be to "find the LCD of the two fractions." In the Fraction Tutor, when multiple paths

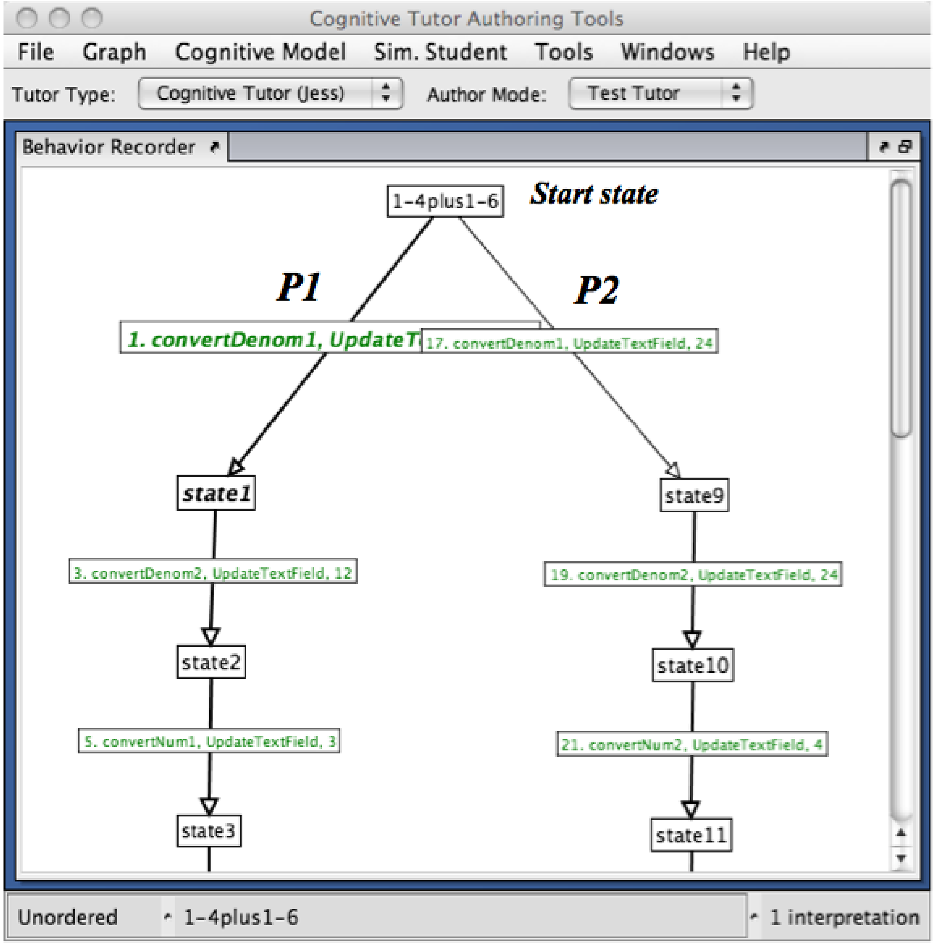

Cognitive Tutor Authoring Tools
File
Graph
Cognitive Model
Sim. Student
Tools
Windows
Help
Tutor Type:
Cognitive Tutor (Jess)
Author Mode:
Test Tutor
Behavior Recorder
1-4plus1-6
Start state
P1
P2
1. convertDenom1, UpdateT
17. convertDenom1, UpdateTextField, 24
state1
state9
3. convertDenom2, UpdateTextField, 12
19. convertDenom2, UpdateTextField, 24
state2
state10
5. convertNum1, UpdateTextField, 3
21. convertNum2, UpdateTextField, 4
state3
state11
Unordered
1-4plus1-6
1 interpretation

Fig. 8. Graph of possible states and paths to solution of the given fraction addition problem. Note that this graph is not complete as it does not allow for all possible orderings of production rules. The current state of the student whose progress is shown in Figure 7 is automatically deduced by the tutor and is shown in bold in the figure above. Note that the $P1$, $P2$, and "start state" were added to the graph and correspond to the productions listed in Section VII-C.

emerge from the current state, the hint to be given is decided based on the "saliency" (a preference value set by the system designer) of each production rule. Recent work has experimented with fully observable Markov Decision Processes to present "optimal hints" to the user based on problem-solving data collected from other students [16], [17].

F. Learning and Knowledge Tracing

Cognitive tutors assume that a student acquires knowledge of a particular production rule through practice. A production rule can only be practiced at those points in the solution graph of a tutoring problem that fulfill the pre-condition of that production rule. When the declarative memory state fulfills the pre-condition ("If"-part) of a particular production rule, then that production is said to have an *opportunity* to fire. At each firing opportunity, the student may learn that production (with some probability) if it is not learned already. If it was already learned, then the student may fire it correctly with some probability $1-s$, where s is a "slipping" parameter (with the same meaning as in our "m independent items" discussion). If it was not learned, then with probability g the student may guess the response corresponding to that production rule. The cognitive tutor keeps track of which productions are learned or not learned by the student by monitoring the student's observations in a process known as *knowledge tracing*.

G. Reduction to m Independent Items

Cognitive tutors typically model the unlearned/learned state of each production rule using exactly the same all-or-nothing model as we described in Section V [35]. Since ACT-R assumes that production rules can be learned independently, teaching of cognitive skills effectively reduces to the m independent items problem. This means that all of the existing literature on the m independent items problem, and all more recently developed POMDP machinery for solving such problems, is relevant for cognitive tutors as well.

H. Selecting the Next Problem

In cognitive tutoring systems, after the student completes a tutoring problem, the tutor must select another problem for the student to tackle. Some cognitive tutors have a fixed threshold for the minimum probability that each production to be taught is in the learned state [35]. Hence, if some productions are below this threshold, then a task should be selected in which the student can gain greater mastery of them.

From the literature, it is unclear how exactly this is performed in practice because important details are left unresolved, and we must assume that heuristic methods are employed. However, one can envision some reasonable strategies. For instance, the tutor might select the problem containing the highest number of production rules whose knowledge by the student is assumed to be below some threshold. Since one problem might be solvable using one of several alternative production rules, another factor to consider is how *likely* a student is to choose a particular production when solving the next problem.

I. Formulation as a POMDP

The cognitive tutoring architecture, whereby a student learns a set of production rules which enable him/her to progress through a sequence of declarative memory states from the starting state to the solution (Figure 8), can be modeled as a POMDP. Doing so allows one to re-formulate both *model tracing* and *knowledge tracing* as a probabilistic inference procedure. Several POMDP formulations are possible; we sketch one below. For simplicity, we omit discussion of the Hint actions that cognitive tutors can execute.

For a given problem domain (e.g., geometry), let the 2^m-dimensional state space representing the learned/unlearned state of m productions be called P. Let D represent the declarative memory state.

In our POMDP formulation we define the state space $\mathcal{S}$ to be $P \times D$; hence, the state at any particular time will be called (p, d). The reward will be defined so that, at the end of the time horizon, the reward is equal to the number of productions that were learned. The action space $\mathcal{A}$ contains a Proceed and several NextProblem actions, corresponding to the set of practice problems contained by the tutor. The Proceed action simply waits for the student to perform some operation within the cognitive tutoring system, e.g., enter the numerator of a fraction into a textbox. Presumably, the tutor will invoke the Proceed action many times as the student progresses from the starting point to the goal of each learning problem. The NextProblem actions reset the declarative memory portion of the state (d) to match the starting condition of some new tutoring problem; it does not affect p, i.e., which production rules are currently learned/unlearned by the student. The observation space $\mathcal{O}$ consists of all possible inputs into the tutor's user interface. It would include all the possible inputs into the various fraction textboxes that might occur when the student fires a particular production rule.

If the declarative memory state d corresponds to an opportunity to fire a production (i.e., that production's pre-condition is fulfilled), then when the tutor invokes the Proceed action, the student may probabilistically change from production state p to p', reflecting the fact that he/she has learned that production. In the event the production fired (by guessing, if the production was not learned, or by not slipping, if the production was learned), then the declarative memory state may also change from d to d'.

The tutor only sees the observations, in the form of inputs into the tutor's graphical user interface. In the Fraction Tutor, these inputs *unambiguously* specify the production rule that fired. However, under the POMDP formulation this restriction is not necessary. Based on the observations it receives, the teaching agent can infer information about the state (p, d) using the belief update step described in Section III.

Since p encompasses the student's knowledge of productions, and d specifies the student's progress along the solution graph, this POMDP formulation allows both tasks of *knowledge tracing* and *model tracing*, which are vital to cognitive tutors, to be computed elegantly as a simple belief update using Equation 2.

Whether the optimal policy for the above POMDP formulation could be tractably computed in practice remains to be investigated. What we wish to point out in this section is that the tasks of knowledge tracing, model tracing, and next-problem selection – which are typically handled separately, and often heuristically, in cognitive tutors – can be integrated in one principled framework. By doing so, and by realizing that the all-or-nothing model of production rule knowledge [35] reduces the teaching problem to the "m independent items" scenarios, one gains access to a wide range of optimization techniques from a rich literature (e.g., stochastic optimal control, reinforcement learning, machine learning) that can be applied in automated teaching domains.

This concludes our treatment of ACT-R, cognitive tutors, and the fundamental teaching scenarios we wished to approach using POMDPs. In the next sections we discuss what we believe to be key issues on the frontier of optimal teaching machines.

VIII. Better Learning Models

Thus far we have assumed that a model of the learner already exists – both its structure (number of states) and parameters. The necessity of a model is argubably a weakness of the POMDP approach, but there are techniques from the machine learning literature which mitigate the modeling issue. Perhaps the most common learner model used in the automated teaching literature is the all-or-nothing model. While elegant in its simplicity, it does not model well-known psychological phenomena such as the Spacing Effect. In general, it is important to have methods for developing and tuning new models.

Assuming a sufficiently large dataset of student observation data is available – e.g., correct/incorrect responses to specific item queries – one could employ Hidden Markov Models to infer the most likely transition and observation probabilities of a finite state model (for a specified fixed number of states k) using the well-known Baum-Welch algorithm. The learned states will not necessarily have intuitive interpretations, but for use in optimal policy planning this need not be a concern.

A. User Dependent Models

It is conceivable that the parameters of a learning model might vary substantially from student to student. For instance, the learning probability p for a particular item might vary between the faster- and slower-learning students within a class. In such cases, it may be beneficial for the teaching machine to infer the learning model *while teaching*.

One method of doing so is to "fold" the parameters of the learning model into the state space of the POMDP. Most simply, this can be performed by discretizing the parameter space into a finite set of values $\mathcal{C}$, and then forming a new state space $\mathcal{S}'$ as the Cartesian product $\mathcal{S} \times \mathcal{C}$. The resultant POMDP will then simultaneously explore the space of parameters of the model, and attempt to exploit the knowledge of the model it has already received, in order to maximize the expected long-term reward.

An alternative approach is to treat the parameters of the model as being drawn from an appropriate Bayesian conjugate distribution, such as the Beta or Dirichlet distribution. Posterior probability estimates of the parameter estimates can be computed at each time step based on the counts of observations received and transitions that occur during the agent-environment interaction. As with the "discretized" approach to parameter exploration described above, this method forms a new state space $\mathcal{S}'$ as the Cartesian product of the original state and the set of all possible counts of observations and/or transitions. What results is a so-called Bayes-Adaptive POMDP [36]. While the new state space is in general enormous, techniques exist to weigh off complexity of the policy computation process with performance of the resulting policy. A simplified version of this approach for optimal teaching was employed even before the advent of the original POMDP formulation [7].

Finally, if one can define a few "protoypical" student models including their parameter values, then a recently developed method [20] to simultaneously infer the correct "student type" and achieve (approximately) optimal learning gains is to perform *policy switching*: Using the observations received so far, one can derive posterior probability estimates that the student matches each prototype. The optimal action can then be sampled, according to those probabilities, from the optimal actions for each prototypical student policy.

IX. Integrating Machine Perception

The models of learning and teaching we have discussed thus far have all but ignored the issue of perception – the only input the teacher receives from the student is a correct or incorrect response (for the all-or-nothing model) or at most a few keystrokes and mouse clicks necessary to enter the answer (for the ACT-R-based cognitive tutors). A human tutor, in contrast, is continuously bombarded with much richer sensor inputs at a much shorter temporal scale and a higher rate of throughput. Relatively little research on automated teaching has examined how to perceive important information about the student not directly related to the answer of the current question, and even less has studied how to use it for control.

As a simple example of why perception is important to teaching, we examine one of the few papers in the cognitive tutoring literature that employed advanced sensors. Gluck, et al [37] augmented an Algebra Tutor with an eye-gaze tracker in order to detect whether particular kinds of errors in problem solving could be predicted from the student's eye gaze. For instance, suppose the student had omitted intermediary steps in his/her calculation of the final answer to an algebra word problem displayed on the screen. If the student erroneously enters the final result into a textbox intended for an intermediary result, then the tutor might think the student had failed to grasp the problem, which would be an incorrect conclusion. However, if the tutor had noticed that the student's eye gaze was fixated on parts of the problem text relevant for the final answer, then it might infer that the student had simply entered his/her response into the wrong box. Being able to distinguish between simple carelessness and a real lack of understanding is important.

Gluck, et al also used the eye gaze tracker to study whether students fixated more on the textual description of the problem, or just on the displayed algebraic equation. They then correlated the more prominent fixation location with probability of solving the problem correctly. They found that focusing on the expression alone was associated with lower error probability than focusing on the problem text alone. Regardless of the particular findings of their study, their work underlines two important lessons about machine perception in teaching: First, good sensors can be used to *disambiguate* two possible causes of the same observation: Was the student sloppy, or did he lack understanding? Did the student leave the room, or is he still working on the problem? Second, sensors, can be used pro-actively to steer students out of undesirable states. If the student's gaze is focused too long on an irrelevant part of the screen, then the student might have "zoned out," and it would be prudent to call this to the student's attention.

A. The Perceptual Vygotsky Problem

While machine perception technology has produced sophisticated sensors in recent years, including eye-gaze trackers [37], facial expression recognition systems [38], [39], and pressure-based posture sensors [40], it is unclear how to use this information in practice. Here we discuss the problem of perceptual teaching machines in the context of the Vygotsky Problem from Section VI-B. Suppose we augmented the Vygotsky teacher with a machine perception capability such as facial expression recognition that triggers sensor events many times per second. How could this information be integrated into the Vygotsky teaching machine, whose intrinsic teaching time scale might be much longer? While this is an open problem, we suggest three alternative approaches that may be useful in real perceptual teaching machines. We call them the Aggregated, Time-driven, and Hierarchical approaches; we discuss each below.

1) Aggregated: One simple approach is to retain the same time scale as was used by the teacher without machine perception. The teacher may be either *event driven*, if for instance the teacher waits indefinitely for the student to issue some response while teaching, or *time driven*, in which case teaching events are always terminated after a fixed duration. In either case, in the aggregated approach, some sufficient statistic of the sensor information, which was presumably received multiple times during a single teaching action, is computed. This statistic is then received by the teacher as an additional dimension of the observation vector.

This architecture does not allow the sensors to interrupt a teaching action. This means, for instance, that even if a student's facial expression clearly indicated at the start of teaching that the student was confused, the confused expression can only be handled at the end of the teaching event (when the observation is issued). Even this limited form of sensing may be better than nothing, however, and requires little change to the existing architecture. Another use of the Aggregated approach is to determine a good prior distribution over the student's learning state: If the facial analysis software indicates, for instance, that the student is very young, then this may suggest the initial level of the student should be low.

2) Time driven: For the Time Driven approach, we assume that the original Vygotsky teaching actions were time driven (or could be converted to be so), and that the time scale of the sensors is shorter than that of teaching. In this approach, the states and actions of the Vygotsky teacher are "sub-divided" so that the time scales of the teacher and the sensors exactly match. If teaching previously advanced the student by 1 level in 10 seconds, and the sensors operate 5 times per second, then the adapted teaching model would advance the student by 0.02 levels every 0.2 seconds so that the teaching and sensor events are synchronized. The state and action spaces would need to be both expanded by a factor of 50. Each individual sensor reading can now be processed in the form of an observation, and computing a sufficient statistic is not necessary. Compared to the Aggregated approach, the Time Driven model may be more responsive since there is no waiting until the end of the teaching action to handle the sensor input. However, scaling down the time scale means the POMDP time horizon becomes longer for the same wall-clock time, and the state space is increased as well. This may pose a tractability problem.

3) Hierarchical: The hierarchical approach attempts to retain the high responsiveness of the Time Driven architecture while reducing the complexity of policy computation, at the expense of full optimality. As in the Time Driven approach, the time scale of the teacher is adjusted to that of the sensor, and the state space is expanded accordingly. But now, instead of modeling levels $1, 1.02, 1.04, \ldots, 2, 2.02, \ldots, m$ with a single "monolithic" POMDP, we divide the teaching problems into two tasks: the "high-level" teacher, responsible for advancing the student from level 1 to 2, 2 to 3, etc.; and the "low-level" teacher, which advances the student by exactly 1 level, divided into fine gradations $(0, 0.02, 0.04, \ldots, 1)$ in accordance with the sensor's intrinsic time scale.

From the standpoint of the high-level POMDP, each teaching action attempts to advance the student from level i to level $i + 1$ by invoking the low-level POMDP as a "black box." The low-level POMDP, on the other hand, knows nothing of the different levels $1, 2, \ldots, m$ – it is myopic and knows only of the fine grained levels necessary to advance the student by exactly one level.

This approach retains the responsiveness of the Time Driven method but sheds much of the complexity since both the high-level and the low-level POMDPs are much smaller in both time horizon and state size than the monolithic POMDP. The disdavantage is that, since the low-level POMDP knows nothing about previous or future levels of the high-level POMDP, it cannot take all possible actions that the monolithic model could; hence, the solution is not truly optimal.

X. Approximate Solution Methods

The two main tractability issues for POMDP-based optimal policy computation are the time horizon (the worst-case time complexity of value iteration is doubly exponential in the time horizon) and the state representation. Both of these issues likely posed considerable difficulty for the early pioneers of optimal control-based teaching in the 1960s and 1970s. More recently approximate solution methods that now exist may partially overcome these challenges, however. If POMDPs are to be used in real-life teaching applications, then exploiting such methods will be important.

Various *point-based* methods of value iteration limit the exponential impact of time horizon by bounding the size of the optimal value function representation to be constant. A smaller representation (quantified in the number of hyperplanes used to represent the value function) results in faster policy computation but higher *regret*, i.e., discrepancy between the achieved and the true optimal value function. All of the experimental results reported in this paper were computed using a point-based approximation.

For mitigating the 2^m state space issue, one may consider avoiding the computation of the optimal value function over beliefs b_t altogether, and instead using a *policy gradient* approach. With policy gradient, the policy π is parameterized with a continuous-valued vector θ. The argument of π need not be a function of b_t – its argument could instead be a sufficient statistic. For the m independent items problem, for example, this might be the m individual probabilities of the learned state for the m independent items. The parameter θ is then tuned using gradient ascent to maximize the value of the policy. Though the resultant policy is only approximately optimal, this method completely avoids the exponential blow-up in state representation.

Finally, by reaching outside the immediate realm of POMDPs into the larger field of reinforcement learning, one can give up on true optimal decision-making and instead learn from observations of expert human teachers; this is the approach of *apprenticeship learning*. While the state is assumed to be visible to the agent, the reward function is assumed to be unknown. A human teacher in the role of the "master" provides examples in the form of state-action "trajectories" of how he/she would act in a particular state. The computer "apprentice" attempts to infer what reward function the human teacher was implicitly optimizing. After inferring the reward function, the optimal policy for this function can then be computed. Apprenticeship learning was used to train a robot teacher that interacts with toddlers in [41].

XI. Conclusion

In this paper we examined the problem of automated teaching from an optimality perspective. We explored three fundamental teaching scenarios – m independent items, items with dependencies, and cognitive skill learning – and showwed in all three cases that the inference and decision-making problems can be handled by the POMDP framework. While the use of POMDPs typically raises questions of tractability, more recently developed solution methods such as point-based approximations and policy gradient approaches may offer help. Finally, we have described possible areas for further research – developing better learner models, and integrating machine perception – and suggested ways of approaching these problems.

We end by re-emphasizing why we approached automated teaching from an optimal control perspective in the first place: By framing teaching as a POMDP, one immediately gains access to a wide range of solution methods, not just from the POMDP literature itself, but also from related and broader fields of stochastic optimal control, reinforcement learning, online learning, and machine learning in general. We believe that this rich literature may hold considerable promise for bringing automated teaching machines to the next level.

References

[1] K. R. Koedinger and J. R. Anderson, "Intelligent tutoring goes to school in the big city," *International Journal of Artificial Intelligence in Education*, vol. 8, pp. 30–43, 1997.

[2] A. Corbett, *User Modeling 2001*. Springer-Verlag, 2001, ch. Cognitive Computer Tutors: Solving the Two-Sigma Problem, pp. 137–147.

[3] B. S. Bloom, "The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring," *Educational Researcher*, vol. 13, no. 6, pp. 4–16, 1984.

[4] J. Mostow, A. Hauptmann, L. Chase, and S. Roth, "Towards a reading coach that listens: Automated detection of oral reading errors," in *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI93)*, 1993.

[5] P. Suppes, "Problems of optimization in learning a list of simple items," in *Human Judgments and Optimality*, M. W. S. II and G. L. Bryan, Eds. John Wiley and Sons, 1964.

[6] R. D. Smallwood, “The analysis of economic teaching strategies for a simple learning model,” *Journal of Mathematical Psychology*, vol. 8, pp. 285–301, 1971.

[7] J. E. Matheson, *Optimum teaching procedures derived from mathematical learning models*. Institute in Engineering-Economic Systems, Stanford University, 1964.

[8] J. H. Laubsch, “A adaptive teaching system for optimal item allocation,” Stanford Institute for Mathematical Studies in the Social Sciences, Tech. Rep., 1969.

[9] R. C. Atkinson, *Human Memory and the Learning Process: Selected Papers of Richard C. Atkinson*. Moscow: Progress Publishing House, 1980, ch. Adaptive instructional systems: Some attempts to optimize the learning process.

[10] ——, “Ingredients for a theory of instruction,” Stanford Institute for Mathematical Studies in the Social Sciences, Tech. Rep., 1972.

[11] J. R. Anderson, *The Architecture of Cognition*. Harvard University Press, 1983.

[12] ——, *Rules of the Mind*. Lawrence Erlbaum Associates, 1993.

[13] J. R. Anderson, C. F. Boyle, and B. J. Reiser, “Intelligent tutoring systems,” *Science*, vol. 228, no. 4698, pp. 456–462, 1985.

[14] C. Conati, A. Gertner, and K. VanLehn, “Using bayesian networks to manage uncertainty in student modeling,” *User Modeling and User-Adapted Interaction*, vol. 12, no. 4, pp. 371–417, 2004.

[15] J. Beck, K. Chang, J. Mostow, and A. Corbett, “Does help help? introducing the bayesian evaluation and assessment methodology,” in *Intelligent Tutoring Systems*, 2008.

[16] Barnes and J. Stamper, “Toward automatic hint generation for logic proof tutoring using historical student data,” in *Intelligent Tutoring Systems*, 2008.

[17] D. Fossati, B. D. Eugenio, S. Ohlsson, C. Brown, L. Chen, and D. Cosejo, “I learn from you, you learn from me: How to make ilist learn from students,” in *The 14th International Conference on Artificial Intelligence in Education*, 2009.

[18] R. C. Murray, K. VanLehn, and J. Mostow, “Looking ahead to select tutorial actions: A decision-theoretic approach,” *International Journal of Artificial Intelligence in Education*, 2004.

[19] H. Gamboa and A. Fred, “Designing intelligent tutoring systems: a bayesian approach,” in *Proceedings of the Ana Fred 3rd International Conference on Enterprise Information Systems*, 2001.

[20] G. Theocharous, R. Beckwith, N. Butko, and M. Philipose, “Tractable pomdp planning algorithms for optimal teaching in spais,” in *IJCAI Workshop on Plan, Activity, and Intent Recognition*, 2009.

[21] R. C. Murray and K. VanLehn, “DT tutor: A decision-theoretic, dynamic approach for optimal selection of tutorial actions,” in *Fifth International Conference on Intelligent Tutoring Systems*, 2000.

[22] J. R. Movellan, “Primer on partially observable markov decision processes: Discrete time discrete state,” Machine Perception Laboratory, Tech. Rep., 2009.

[23] G. H. Bower, “Application of a model to paired-associate learning,” *Psychometrika*, vol. 26, no. 3, 1961.

[24] R. R. Bush and S. H. Sternberg, “A single-operator model,” in *Studies in Mathematical Learning Theory*, R. R. Bush and W. K. Estes, Eds. Stanford Mathematical Studies in the Social Sciences, Stanford University Press, 1959.

[25] H. A. Bernbach, “A forgetting model for paired-associate learning,” *Journal of Mathematical Psychology*, vol. 2, pp. 128–144, 1965.

[26] W. Karush and R. Dear, “Optimal strategy for item presentation in a learning process,” *Management Science,*, vol. 13, no. 11, pp. 773–785, 1967.

[27] N. J. Cepeda, N. Coburn, D. Rohrer, J. T. Wixted, M. C. Mozer, and H. Pashler, “Optimizing distributed pratice: Theoretical analysis and practical implications,” *Experimental Psychology 2009*, vol. 56, no. 4, 2009.

[28] H. Ebbinghaus, *Memory: A contribution to experimental psychology (‘De subjecto vetustissimo novissimam promovemus scientiam’)*. University of Berlin, 1885.

[29] Wikipedia, 2009, http://en.wikipedia.org/wiki/Zone_of_proximal_development.

[30] J. R. Anderson, *Adaptive Character of Thought*. Lawrence Erlbaum Associates, 1993.

[31] K. VanLehn, C. Lynch, K. Schultz, J. Shapiro, R. Shelby, and L. Taylor, “The andes physics tutoring system: Lessons learned,” *International Journal of Artificial Intelligence and Education*, vol. 15, no. 3, pp. 147–204, 2005.

[32] I. Carnegie Learning, 2009, www.carnegielearning.com.

[33] PACT Center at Carnegie Mellon University, “Cognitive tutor authoring tools,” 2009, http://ctat.pact.cs.cmu.edu/index.php?id=software.

[34] P. I. Pavlik and J. R. Anderson, “Using a model to compute the optimal schedule of practice,” *Journal of Experimental Psychology: Applied*, vol. 14, no. 2, pp. 101–117, 2008.

[35] A. T. Corbett and J. R. Anderson, “Student modeling and mastery learning in a computer-based programming tutor,” in *Proceedings of the Second International Conference on Intelligent Tutoring Systems*, 1992.

[36] S. Ross, B. Chaib-draa, and J. Pineau, “Bayes-adaptive POMDPs,” in *Advances in Neural Information Processing Systems*, 2007.

[37] K. A. Gluck, J. R. Anderson, and S. A. Douglass, “Broader bandwidth in student modeling: What if its were “eye”TS?” in *Intelligent Tutoring Systems*, 2000.

[38] M. Bartlett, G. Littlewort, M. Frank, C. Lainscsek, I. Fasel, and J. Movellan, “Automatic recognition of facial actions in spontaneous expressions,” *Journal of Multimedia*, 2006.

[39] J. Whitehill, M. Bartlett, and J. R. Movellan, “Automatic facial expression recognition for intelligent tutoring systems,” in *Proceedings of the CVPR 2008 Workshop on Human Communicative Behavior Analysis*, 2008.

[40] S. D’Mello, R. Picard, and A. Graesser, “Towards an affect-sensitive autotutor,” *IEEE Intelligent Systems, Special issue on Intelligent Educational Systems*, vol. 22, no. 4, 2007.

[41] P. Ruvolo, J. Whitehill, M. Virnes, and J. Movellan, “Building a more effective teaching robot using apprenticeship learning,” in *Proceedings of the International Conference on Development and Learning*, 2008.